PATENT
INTEL/17225

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant: **Tian et al.** Serial No.: 10/677,414 For: Methods and Apparatus for Reducing Memory Latency in a Software Application Filed: October 2, 2003 Group Art Unit: 2193 Examiner: Todd D. Ingberg	I hereby certify that this paper is being deposited electronically with the United States Patent and Trademark Office on this date: **July 9, 2007** /Michael W. Zimmerman/ Michael W. Zimmerman Registration No. 57,993 Agent for Applicants

RESPONSE TO THE OFFICE ACTION DATED APRIL 9, 2007

Mail Stop Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

In response to the Office action dated April 9, 2007, please enter the following amendments and consider the following remarks:

The Status of the Claims is reflected in the listing of claims that begins on page 2 of this paper.

Remarks begin on page 10 of this paper.

This listing of claims will replace all prior versions, and listings, of claims in the application:

The Status of the Claims

1. (Original) A method of reducing memory latency in a software application, the method comprising:

analyzing the software application and determining a first area of software instructions that encounters a cache miss;

generating a helper thread;

generating a first set of compiler-runtime instructions and inserting the first set of compiler-runtime instructions in a main thread;

generating a second set of compiler-runtime instructions and inserting the second set of compiler-runtime instructions in the helper thread; and

inserting a counting mechanism in the main thread and the helper thread, the counting mechanism being structured to coordinate relative execution points of the main thread and the helper thread.

2. (Original) A method as defined in claim 1, further comprising analyzing the software application and determining a second area of software instructions which encounters a memory load latency.

3. (Original) A method as defined in claim 2, wherein the first area of software instructions is different than the second area of software

instructions.

4. (Original) A method as defined in claim 2, wherein the first area of software instructions comprises the second area of software instructions.

5. (Original) A method as defined in claim 2, wherein analyzing the software application comprises:

measuring cache miss rates associated with the software application using a performance analysis tool;

measuring memory load latencies associated with the software application using the performance analysis tool;

reporting the first area of software instructions which encounters the cache miss to a compiler; and

reporting the second area of software instructions which encounters the memory load latency.

6. (Original) A method as defined in claim 1, wherein generating a helper thread includes generating a thread graph.

7. (Original) A method as defined in claim 6, wherein the thread graph presents a data structure that represents a relationship between the main thread and the helper thread.

8. (Original) A method as defined in claim 6, wherein the thread graph facilitates code reuse.

9. (Original) A method as defined in claim 1, wherein at a least part of the first set of compiler-runtime instructions comprises at least a part of the second set of compiler-runtime instructions.

10. (Original) A method as defined in claim 1, wherein the first set of compiler-runtime instructions inserted in the main thread comprises instructions to spawn the helper thread, terminate the helper thread, and coordinate execution of the helper thread and the main thread.

11. (Original) A method as defined in claim 1, wherein the second set of compiler-runtime instructions inserted in the helper thread comprises instructions to coordinate execution of the helper thread and the main thread.

12. (Original) A method as defined in claim 1, wherein the counting mechanism comprises a software counter.

13. (Original) A method as defined in claim 12, wherein at least one of the first set of compiler-runtime instructions and the second set of compiler-runtime instructions include instructions to control execution rates of the helper thread based on a value associated with the software counter.

14. (Original) A method as defined in claim 12, wherein at least one of the first set of compiler-runtime instructions and the second set of compiler-runtime instructions include instructions to control execution rates of the main thread based on a value associated with the software counter.

15. (Original) A method as defined in claim 14, wherein the compiler-runtime instructions to control execution rates comprises a delay instruction, a catch up instruction and an instruction to force execution.

16. (Original) A system to reduce memory latency, the system comprising:

a processor;

a memory operatively coupled to the processor:

the memory storing a software tool structured to identify a code region in an application program that suffers from a data cache miss;

a compiler operatively coupled to the software tool, the compiler being structured to receive information from the software tool and to generate a helper thread;

a set of compiler-runtime instructions to be generated and inserted in the application program to manage the helper thread and to manage a main thread; and

a counting mechanism for insertion in the main thread and the helper thread to facilitate coordination of execution points associated with the helper thread and the main thread.

17. (Original) A system as defined in claim 16, wherein the software tool comprises a VTune™ Performance Anaylyzer.

18. (Original) A system as defined in claim 16, wherein the information the compiler receives from the software tool comprises data cache miss rates associated with the identified code region.

19. (Original) A system as defined in claim 16, wherein the information the compiler receives from the software tool comprises memory load latency times associated with the identified code region.

20. (Original) A system as defined in claim 16, wherein the helper thread is structured to prefetch variables contained in the identified code region.

21. (Original) A system as defined in claim 16, wherein the set of compiler-runtime instructions comprises instructions to create the helper thread, terminate the helper thread, delay execution of the helper thread, and activate the helper thread.

22. (Original) A system as defined in claim 16, wherein the set of compiler-runtime instructions comprises instructions to coordinate execution of the helper thread and the main thread.

23. (Original) A machine readable medium storing instructions to cause a machine to:

analyze a software application including a main thread;

identify a code region in the software application;

generate a helper thread;

generate and insert a first set of compiler-runtime instructions in the main thread to manage the helper thread and the main thread;

generate and insert a second set of compiler-runtime instructions in the helper thread to manage the helper thread and the main thread; and

manage execution points of the helper thread and the main thread.

24. (Original) A machine readable medium as defined in claim 22, wherein the stored instructions are structured to cause the machine to identify the code region based on cache miss rates.

25. (Original) A machine readable medium as defined in claim 22, wherein the stored instructions are structured to cause the machine to identify the code region based on memory load latencies.

26. (Original) A machine readable medium as defined in claim 22, wherein the stored instructions are structured to cause the machine to generate the helper thread to prefetch instructions within the identified code region.

27. (Original) A machine readable medium as defined in claim 22, wherein the stored instructions are structured to cause the machine to generate compiler-runtime instructions to spawn the helper thread, terminate the helper thread, coordinate execution of the helper thread and the main thread.

28. (Original) A machine readable medium as defined in claim 22, wherein the stored instructions are structured to cause the machine to manage the execution of the main thread and the helper thread by inserting a first portion of a counting mechanism in the main thread and a second portion of a counting mechanism in the helper thread.

29. (Original) An apparatus to reduce memory latency, the apparatus comprising:

a software tool structured to identify a code region in an application program that suffers from a data cache miss;

a compiler operatively coupled to the software tool, the compiler being structured to receive information from the software tool and to generate a helper thread;

a set of compiler-runtime instructions to be generated and inserted in the application program to manage the helper thread and to manage a main thread; and

a counting mechanism for insertion in the main thread and the helper thread to facilitate coordination of execution points associated with the helper thread and the main thread.

30. (Original) An apparatus as defined in claim 29, wherein the information the compiler receives from the software tool comprises data cache miss rates associated with the identified code region.

31. (Original) An apparatus as defined in claim 29, wherein the information the compiler receives from the software tool comprises memory load latency times associated with the identified code region.

32. (Original) An apparatus as defined in claim 29, wherein the helper thread is structured to prefetch variables contained in the identified code region.

33. (Original) An apparatus as defined in claim 29, wherein the set of compiler-runtime instructions comprises instructions to create the helper thread, terminate the helper thread, delay execution of the helper thread, and activate the helper thread.

34. (Original) An apparatus as defined in claim 29, wherein the set of compiler-runtime instructions comprises instructions to coordinate execution of the helper thread and the main thread.